日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００３年　７月２２日

出願番号
Application Number: 特願２００３－１９９９５０

[ST. 10/C]: [ＪＰ２００３－１９９９５０]

出願人
Applicant(s): 株式会社リコー

２００３年　８月　４日

特許庁長官
Commissioner,
Japan Patent Office 今井康夫

出証番号　出証特２００３－３０６２１７６

【書類名】　特許願
【整理番号】　0305247
【提出日】　平成15年 7月22日
【あて先】　特許庁長官　今井　康夫　殿
【国際特許分類】　G03G 21/00　370
【発明の名称】　情報処理装置およびプログラム生成方法
【請求項の数】　22
【発明者】
　【住所又は居所】　東京都大田区中馬込１丁目３番６号　株式会社リコー内
　【氏名】　秋吉　邦洋
【特許出願人】
　【識別番号】　000006747
　【氏名又は名称】　株式会社リコー
【代理人】
　【識別番号】　100070150
　【弁理士】
　【氏名又は名称】　伊東　忠彦
【先の出願に基づく優先権主張】
　【出願番号】　特願2002-276532
　【出願日】　平成14年 9月24日
【手数料の表示】
　【予納台帳番号】　002989
　【納付金額】　21,000円
【提出物件の目録】
　【物件名】　明細書　1
　【物件名】　図面　1
　【物件名】　要約書　1
　【包括委任状番号】　9911477
【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　情報処理装置およびプログラム生成方法

【特許請求の範囲】

【請求項１】　プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムに実装され、各モジュールで共用する共用変数に対し前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリ

を備えたことを特徴とする情報処理装置。

【請求項２】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする請求項１に記載の情報処理装置。

【請求項３】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする請求項１または２に記載の情報処理装置。

【請求項４】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする請求項１～３のいずれか一つに記載の情報処理装置。

【請求項５】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする請求項１～４のいずれか一つに記載の情報処理装置。

【請求項６】　前記情報処理装置は、画像形成処理を実行する画像形成装置である請求項１～５のいずれか一つに記載の情報処理装置。

【請求項７】　画像形成処理で使用されるハードウェア資源と、

前記画像形成装置のアプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通に

必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、

前記プログラムは、前記アプリケーションであり、前記ユーティリティライブラリは、当該アプリケーションに結合されていることを特徴とする請求項６に記載の情報処理装置。

【請求項８】　プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムのオブジェクトファイルと、各モジュールで共用する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリとを結合するリンクステップ

を含むことを特徴とするプログラム生成方法。

【請求項９】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする請求項８に記載のプログラム生成方法。

【請求項１０】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする請求項８または９に記載のプログラム生成方法。

【請求項１１】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする請求項８～１０のいずれか一つに記載のプログラム生成方法。

【請求項１２】　前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする請求項８～１１のいずれか一つに記載のプログラム生成方法。

【請求項１３】　前記プログラム生成方法により生成されるプログラムは、画像形成装置で実行されるプログラムである請求項８～１２のいずれか一つに記載のプログラム生成方法。

【請求項１４】　前記画像形成装置は、

画像形成処理で使用されるハードウェア資源と、

前記画像形成装置のアプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスとを備え、

前記プログラムは前記アプリケーションであることを特徴とする請求項１３に記載のプログラム生成方法。

【請求項１５】　プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルであって、

当該状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムに実装され、各モジュールで共用する共用変数に対し前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えた情報処理装置において用いられる状態遷移モデル。

【請求項１６】　プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムのオブジェクトファイルと、各モジュールで共用する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリとを結合することにより生成されるプログラム。

【請求項１７】　プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを、情報処理装置に生成させるプログラムであって、情報処理装置に、

メッセージの受信に応じて各モジュールで共用する共用変数を前記モジュール

ごとに別個の領域に割り当てる手順と、

イベント関数テーブルを参照することにより前記メッセージに応じたイベント関数を実行する手順と

を実行させるプログラム。

【請求項１８】　前記状態の遷移が生じたときに行う処理を定めた入場関数を更に実行させる請求項１７に記載のプログラム。

【請求項１９】　前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を更に実行させる請求項１７または１８に記載のプログラム。

【請求項２０】　前記情報処理装置は、画像形成処理を実行する画像形成装置である請求項１７～１９のいずれか一つに記載のプログラム。

【請求項２１】　前記画像形成装置は、

画像形成処理で使用されるハードウェア資源と、

前記画像形成装置のアプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、

前記プログラムは、前記アプリケーションである請求項２０に記載のプログラム。

【請求項２２】　請求項１６～２１のいずれか一つに記載のプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

この発明は、コピー、プリンタ、スキャナおよびファクシミリなどの画像形成処理にかかるユーザサービスを提供し、状態遷移モデルに基づいてアプリケーションを開発するための関数が登録されたユーティリティライブラリを備えた画像形成装置およびアプリケーション生成方法に関するものである。

【０００２】

【従来の技術】

近年では、プリンタ、コピー、ファクシミリ、スキャナなどの各装置の機能を１つの筐体内に収納した画像形成装置（以下、「複合機」という。）が知られている。この複合機は、１つの筐体内に表示部、印刷部および撮像部などを設けるとともに、プリンタ、コピーおよびファクシミリ装置にそれぞれ対応した３種類のソフトウェア（アプリケーション）を設け、これらのソフトウェアを切り替えることによって、当該装置をプリンタ、コピー、スキャナまたはファクシミリ装置として動作させるものである。

【０００３】

このような従来の複合機では、複合機上で動作するアプリケーションの開発は、複合機の開発元で一括して行い、開発されたアプリケーションをＲＯＭなどに焼き付けた状態で複合機を出荷し、出荷後は新たなアプリケーションの追加を行わないのが一般的である。

【０００４】

ところで、このような従来の複合機では、プリンタ、コピー、スキャナおよびファクシミリ装置に対応するソフトウェアをそれぞれ別個に設けているため、各ソフトウェアの開発に多大の時間を要する。このため、出願人は、表示部、印刷部および撮像部などの画像形成処理で使用されるハードウェア資源を有し、プリンタ、コピーまたはファクシミリなどの各ユーザサービスにそれぞれ固有の処理を行うアプリケーションを複数搭載し、これらのアプリケーションとハードウェア資源との間に介在して、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行う各種コントロールサービスからなるプラットホームを含む画像形成装置（複合機）を発明した（例えば、特許文献１参照）。

【０００５】

【特許文献１】

特開２００２－８２８０６号公報

【０００６】

【発明が解決しようとする課題】

このような新規な複合機は、アプリケーションの少なくとも２つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行うプラットホームを備えた構成とすることによって、ソフトウェア開発の効率化を図るとともに、装置全体としての生産性を向上させることができる点で優れている。

【０００７】

一方、このような新規な複合機では、アプリケーションと、ハードウェア資源にアクセスするような開発が難しい処理を行うコントロールサービスとが別個に設けられているため、複合機の出荷後にユーザもしくは第三者であるサードベンダが新規な外部アプリケーションを開発して複合機に搭載することができる構成となっている。

【０００８】

このため、このような複合機では、複数のサードベンダによってそれぞれ別個に開発されたアプリケーションが複数搭載されて同時に動作する場合がある。また、一つのサードベンダが開発した一つのアプリケーションであっても、そのアプリケーション内部の複数のモジュールを別々の部署で自由に開発して一つのアプリケーションとして統合して提供する場合も考えられる。

【０００９】

このように、アプリケーションの開発を複合機の開発メーカ以外のサードベンダ等に委ねた場合、各サードベンダが独自の手法でアプリケーション開発を行うこととなってしまうため、開発されたアプリケーション内部で障害が発生したり、出荷時に予め搭載されている既存のアプリケーションや複合機のシステムに悪影響を与えてしまうおそれがある。特に、アプリケーションプログラムで使用される共用変数や複合機独自のイベントについて全くその使用方法をサードベンダに委ねてしまうと、ソフトウェア障害によって複合機のシステムが不安定になってしまうおそれが高い。このような問題は、画像形成装置上で動作するアプリケーションのみならず、一般的な情報処理装置上で動作するアプリケーションにとっても問題となることである。

【００１０】

この発明は上記に鑑みてなされたもので、システムの安定性を保持しつつアプ

リケーションの追加を容易に行うことができる情報処理装置およびアプリケーション生成方法を得ることを目的とする。

【００１１】

【課題を解決するための手段】

上記目的を達成するため、請求項１にかかる発明は、プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムに実装され、各モジュールで共用する共用変数に対し前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えたことを特徴とする情報処理装置である。

【００１２】

この請求項１の発明によれば、プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の前記状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムに実装され、各モジュールで共有する共用変数を前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えているので、プログラムを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、情報処理装置に新規アプリケーションを搭載可能としながらも情報処理装置の安定性を維持することができる。

【００１３】

また、請求項２にかかる発明は、請求項１に記載の情報処理装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることを特徴とする。

【００１４】

この請求項２の発明によれば、状態遷移モデル起動関数は、前記共用変数を、前記モジュールごとのスタック領域で指定された領域に割り当てることで、関数呼び出しの際に使用されるスタック領域を利用してモジュールが呼び出しの際に効率的に共用変数の割り当てを行うことができる。

【００１５】

また、請求項３にかかる発明は、請求項１または２に記載の画像形成装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることを特徴とする。

【００１６】

この請求項３の発明によれば、状態遷移モデル起動関数は、スレッド単位で共用される共用変数を前記スレッドごとに別個の領域に割り当てることで、プログラムのプロセス内部で動作するスレッド内部で共用変数を安全に使用することができ、変数の値に起因する障害を回避できる。

【００１７】

また、請求項４にかかる発明は、請求項１～３のいずれか一つに記載の情報処理装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することを特徴とする。

【００１８】

この請求項４の発明によれば、状態遷移モデル起動関数は、前記状態の遷移が生じたときに行う処理を定めた入場関数を実行することで、モジュール呼び出しの際に定型的に行われる処理を実行することができ、プログラム開発を効率的に行える。

【００１９】

また、請求項５にかかる発明は、請求項１～４のいずれか一つに記載の情報処理装置において、前記ユーティリティライブラリの前記状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することを特徴とする。

【００２０】

この請求項５の発明によれば、状態遷移モデル起動関数は、前記状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行することで、現在の状態とは無関係に生じるシステム関連のイベントなどに他のプログラムに悪影響を与えずに対処することができ、画像形成装置の安定性を向上させることができる。

【００２１】

また、請求項６にかかる発明は、請求項１～５のいずれか一つの記載において、前記情報処理装置は、画像形成処理を実行する画像形成装置であるとするものである。

【００２２】

また、請求項７にかかる発明は、請求項６の画像形成装置は、画像形成処理で使用されるハードウェア資源と、前記画像形成装置のアプリケーションと前記ハードウェア資源との間に介在し、ユーザサービスを提供する際に、アプリケーションの少なくとも２つが共通に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスと、をさらに備え、前記プログラムは、前記アプリケーションであり、前記ユーティリティライブラリは、当該アプリケーションに結合されていることを特徴とするものである。

この請求項７の発明によれば、前記ユーティリティライブラリは、前記アプリケーションに結合されているので、アプリケーションのソフトウェア開発をユーザサービスに固有の部分だけで行うことができ、ソフトウェア開発の労力軽減を図ることができる。

【００２３】

請求項９～１４にかかる発明は、上記のプログラムの生成方法の発明であり、この発明によっても上記と同様の効果を奏する。

【００２４】

また、請求項１５にかかる発明は、前記情報処理装置における状態遷移モデルの発明であり、このように状態遷移モデルを用いることにより、上記と同様の効果を奏する。請求項１６～２１に記載の発明は、上記情報処理装置で実行される

プログラムの発明であり、請求項２２に記載の発明は、当該プログラムを記録した記録媒体の発明である。

【００２５】

【発明の実施の形態】

以下に添付図面を参照して、この発明にかかる情報処理装置およびアプリケーション生成方法の好適な実施の形態を詳細に説明する。以下、情報処理装置の例として画像形成装置について説明する。

（実施の形態１）

図１は、この発明の実施の形態１である画像形成装置（以下、「複合機」という。）の構成を示すブロック図である。

【００２６】

図１に示すように、複合機１００は、白黒レーザプリンタ（B&W LP）１０１と、カラーレーザプリンタ（Color LP）１０２とハードディスク装置（ＨＤＤ）１０３と、ファクシミリ、メモリなどのハードウェアリソース１０４を有するとともに、プラットホーム１２０と、アプリケーション１３０とから構成されるソフトウェア群１１０とを備えている。

【００２７】

プラットホーム１２０は、アプリケーションからの処理要求を解釈してハードウェア資源の獲得要求を発生させるコントロールサービスと、一または複数のハードウェア資源の管理を行い、コントロールサービスからの獲得要求を調停するシステムリソースマネージャ（ＳＲＭ）１２３と、汎用ＯＳ１２１とを有する。

【００２８】

コントロールサービスは、複数のサービスモジュールから形成され、ＳＣＳ（システムコントロールサービス）１２２と、ＥＣＳ（エンジンコントロールサービス）１２４と、ＭＣＳ（メモリコントロールサービス）１２５と、ＯＣＳ（オペレーションパネルコントロールサービス）１２６と、ＦＣＳ（ファックスコントロールサービス）１２７と、ＮＣＳ（ネットワークコントロールサービス）１２８とから構成される。このプラットホーム１２０は、あらかじめ定義された関数により前記アプリケーション１３０から処理要求を受信可能とするアプリケー

ションプログラムインタフェース（ＡＰＩ）を有する。

【００２９】

汎用ＯＳ１２１は、ＵＮＩＸ（登録商標）などの汎用オペレーティングシステムであり、プラットホーム１２０並びにアプリケーション１３０の各ソフトウェアをそれぞれプロセスとして並列実行する。

【００３０】

ＳＲＭ１２３のプロセスは、ＳＣＳ１２２とともにシステムの制御およびリソースの管理を行うものである。ＳＲＭ１２３のプロセスは、スキャナ部やプリンタ部などのエンジン、メモリ、ＨＤＤファイル、ホストＩ／Ｏ（セントロＩ／Ｆ、ネットワークＩ／Ｆ、ＩＥＥＥ１３９４　Ｉ／Ｆ、ＲＳ２３２Ｃ　Ｉ／Ｆなど）のハードウェア資源を利用する上位層からの要求にしたがって調停を行い、実行制御する。

【００３１】

具体的には、このＳＲＭ１２３は、要求されたハードウェア資源が利用可能であるか（他の要求により利用されていないかどうか）を判断し、利用可能であれば要求されたハードウェア資源が利用可能である旨を上位層に伝える。また、ＳＲＭ１２３は、上位層からの要求に対してハードウェア資源の利用スケジューリングを行い、要求内容（例えば、プリンタエンジンにより紙搬送と作像動作、メモリ確保、ファイル生成など）を直接実施している。

【００３２】

ＳＣＳ１２２のプロセスは、アプリ管理、操作部制御、システム画面表示、ＬＥＤ表示、リソース管理、割り込みアプリ制御などを行う。

【００３３】

ＥＣＳ１２４のプロセスは、白黒レーザプリンタ（B&W LP）１０１、カラーレーザプリンタ（Color LP）１０２、ＨＤＤ１０３、スキャナ、ファクシミリなどからなるハードウェアリソース１０４のエンジンの制御を行う。

【００３４】

ＭＣＳ１２５のプロセスは、画像メモリの取得および解放、ハードディスク装置（ＨＤＤ）の利用、画像データの圧縮および伸張などを行う。

【００３５】

ＦＣＳ１２７のプロセスは、システムコントローラの各アプリ層からＰＳＴＮ／ＩＳＤＮ網を利用したファクシミリ送受信、ＢＫＭ（バックアップＳＲＡＭ）で管理されている各種ファクシミリデータの登録／引用、ファクシミリ読みとり、ファクシミリ受信印刷、融合送受信を行うためのＡＰＩを提供する。

【００３６】

ＮＣＳ１２８のプロセスは、ネットワークＩ／Ｏを必要とするアプリケーションに対して共通に利用できるサービスを提供するためのプロセスであり、ネットワーク側から各プロトコルによって受信したデータを各アプリケーションに振り分けたり、アプリケーションからデータをネットワーク側に送信する際の仲介を行う。

【００３７】

ＯＣＳ１２６のプロセスは、オペレータ（ユーザ）と本体制御間の情報伝達手段となるオペレーションパネル（操作パネル）の制御を行う。ＯＣＳ１２６は、オペレーションパネルからキー押下（またはタッチ操作）をキーイベントとして取得し、取得したキーに対応したキーイベント関数をＳＣＳ１２２に送信するＯＣＳプロセスである。また、オペレーションパネルの操作表示部に対する各種画面を描画出力やその他オペレーションパネルに対する制御は、ＯＣＳ関数ライブラリに登録されている描画関数等の各種関数をアプリケーション１３０またはコントロールサービスから呼び出すことにより行われる。このＯＣＳ関数ライブラリは、アプリケーション１３０およびコントロールサービスの各モジュールに動的にリンクされている。なお、ＯＣＳ１２６のすべてをプロセスとして動作させるように構成しても良く、あるいはＯＣＳ１２６のすべてをＯＣＳ関数ライブラリとして構成しても良い。

【００３８】

アプリケーション１３０は、ページ記述言語（ＰＤＬ）、ＰＣＬおよびポストスクリプト（ＰＳ）を有するプリンタ用のアプリケーションであるプリンタアプリ１１１と、コピー用アプリケーションであるコピーアプリ１１２と、ファクシミリ用アプリケーションであるファックスアプリ１１３と、スキャナ用アプリケ

ーションであるスキャナアプリ１１４と、ネットワークファイル用アプリケーションであるネットファイルアプリ１１５と、工程検査用アプリケーションである工程検査アプリ１１６と、サードベンダなどの第三者が開発した外部アプリ１１７とを有している。外部アプリ１１７は、状態遷移モデルに従った開発手法により生成されており、状態遷移モデル起動関数が登録されたユーティリティライブラリをリンクしている。

【００３９】

アプリケーション１３０の各プロセス、コントロールサービスの各プロセスは、関数呼び出しとその戻り値送信およびメッセージの送受信によってプロセス間通信を行いながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを実現している。

【００４０】

このように、実施の形態１にかかる複合機１００には、複数のアプリケーション１３０および複数のコントロールサービスが存在し、いずれもプロセスとして動作している。そして、これらの各プロセス内部には、一または複数のスレッドが生成されて、スレッド単位の並列実行が行われる。そして、コントロールサービスがアプリケーション１３０に対し共通サービスを提供しており、このため、これらの多数のプロセスが並列動作、およびスレッドの並列動作を行って互いにプロセス間通信を行って協調動作をしながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを提供するようになっている。

【００４１】

また、複合機１００には、複合機１００の顧客、サードベンダなどの第三者がコントロールサービス層の上のアプリケーション層に外部アプリを開発して搭載することが可能となっている。

【００４２】

なお、実施の形態１にかかる複合機１００では、複数のアプリケーション１３０のプロセスと複数のコントロールサービスのプロセスとが動作しているが、アプリケーション１３０とコントロールサービスのプロセスがそれぞれ単一の構成

とすることも可能である。また、各アプリケーション１３０は、アプリケーションごとに追加または削除することができる。

【００４３】

図２に複合機１００のハードウェア構成例を示す。

【００４４】

複合機１００は、コントローラ１６０と、オペレーションパネル１７５と、ファックスコントロールユニット（ＦＣＵ）１７６と、プリンタ等の画像形成処理に特有のハードウェア資源であるエンジン部１７７とを含む。コントローラ１６０は、ＣＰＵ１６１と、システムメモリ１６２と、ノースブリッジ（ＮＢ）１６３と、サウスブリッジ（ＳＢ）１６４と、ＡＳＩＣ１６６と、ローカルメモリ１６７と、ＨＤＤ１６８と、ネットワークインターフェースカード（ＮＩＣ）１６９と、ＳＤカード用スロット１７０と、ＵＳＢデバイス１７１と、ＩＥＥＥ１３９４デバイス１７２と、セントロニクス１７３とを含む。なお、メモリ１６２、１６７はＲＡＭ、ＲＯＭ等を含む。ＦＣＵ１７６およびエンジン部１７７は、コントローラ１６０のＡＳＩＣ１６６にＰＣＩバス１７８で接続されている。

【００４５】

ＣＰＵ１６１が、複合機１００にインストールされるアプリケーション、コントロールサービス等のプログラムを、メモリから読み出して実行する。

【００４６】

次に、外部アプリ１１７が実行されている場合のプロセス内部の構造について説明する。図３は、外部アプリ１１７のプロセス内部の構造を示す説明図である。

【００４７】

外部アプリ１１７のプロセスは、複数のスレッドが並列実行されたマルチスレッド環境となっている。図３に示すように、外部アプリ１１７のプロセスには、メインスレッド２１０とイメージライブラリスレッド２００が並列実行されている。メインスレッド２１０は、外部アプリ１１７の主要な処理を実行するスレッドであり、このメインスレッド２１０がイメージライブラリスレッド２００を呼び出すことにより、コントロールサービス１４０やＶＡＳ（仮想アプリケーショ

ンサービス）などの関数呼び出しとその戻り値の受信、メッセージの送受信を実現している。

【００４８】

また、外部アプリ１１７のプロセスには、サブスレッド２２０、メールボックス２１１，２１２，２１４、イベント取得ハンドラ（スレッド）２１３が存在している。メインスレッド２１０には、状態遷移モデル起動関数が登録されたユーティリティライブラリ２１５が実装されている。

【００４９】

メールボックス２１１は、メインスレッド２１０とイメージライブラリスレッド２００およびイベント取得ハンドラ２１３との通信に使用される。メールボックス２１２は、メインスレッド２１０とイメージライブラリスレッド２００の通信に使用される。イベント取得ハンドラ２１３は、ＥＣＳ１２４，ＭＣＳ１２５などのコントロールサービスまたはＶＡＳのプロセスからのイベントメッセージを受信し、メールボックス２１１に書き込むものである。

【００５０】

メインスレッド２１０がコントロールサービスなどにアクセスするためには、メインスレッド３１０がメールボックス２１１，２１２，２１４を開設した後、イメージライブラリスレッドを起動する。

【００５１】

メインスレッド３１０は、イメージライブラリスレッド２００に対する関数呼び出しのコマンドを、送信元情報がＭＢ＿ＲＥＱＵＥＳＴとしてメールボックス２１１に入れる。また、コントロールサービス１４０などからのイベントメッセージも、送信元情報ＭＢ＿ＣＳのメールとしてメールボックス２１１に入る。

【００５２】

イメージライブラリスレッド２００は、メールボックス２１１に配信されるメールをｒｃｖ＿ｍｓｇ（）関数によって取得する。イメージライブラリスレッド２００は、取得したメールに含まれる送信元情報とイベントまたは関数情報から、対応する関数の検索を行い、対応する関数が検索された場合にその関数を実行する。一方、対応する関数が検索されなかった場合には何も処理は行わない。

【００５３】

取得したメールは対応する関数の実行に関係なく、メールボックス２１２に転送される。また、イメージライブラリスレッド２００は、メインスレッドに対して、エラー情報、実行結果、途中経過などの通知を送信元情報ＭＢ＿ＩＭＡＧＥのメールとして、メールボックス２１２に入れる。また、メインスレッド２１０は、サブスレッド２２０とメールボックス２１４，２１２を利用して同様にメッセージの送受信を行う。

【００５４】

また、ユーティリティライブラリ２１５およびイメージライブラリは、複合機のアプリケーション開発のためのソフトウェア開発キット（ＳＤＫ：Ｓｏｆｔｗａｒｅ　Ｄｅｖｅｌｏｐｍｅｎｔ　Ｋｉｔ）などの開発用ツールキットの一部または全部として、ＣＤ－ＲＯＭまたはＦＤなどの記憶媒体にインストール可能な形式のファイルで提供される。また、ユーティリティライブラリ２１５およびイメージライブラリを、Ｗｅｂサイトからのダウンロード等、ネットワーク経由で取得可能な方法で提供するようにしても良い。

【００５５】

このように提供された上記ユーティリティライブラリ２１５およびイメージライブラリを用いて外部アプリ１１７（複合機１００の出荷後に第三者によって開発されて搭載されるアプリ）を生成するには、外部アプリのソースファイルをコンパイラによってコンパイルしてオブジェクトファイルを生成する。そして、リンカによって、生成されたオブジェクトファイルと、上記の方法で提供されたユーティリティライブラリ２１５およびイメージライブラリとをリンクして、外部アプリの実行形式ファイルを生成する。この外部アプリの実行形式ファイルをフラッシュカードなどに格納して、複合機１００において、フラッシュカードから外部アプリをインストールすることにより、ユーティリティライブラリ２１５およびイメージライブラリを実装した外部アプリが複合機に搭載される。また、外部アプリをネットワークを介して外部のサーバからインストールすることもできる。

【００５６】

次に、外部アプリ１１７が採用している状態遷移モデルについて説明する。図４は、外部アプリ１１７が状態１と状態２を有する場合の状態遷移の説明図である。図４に示すように、外部アプリ１１７のメインスレッド２１０が、状態１および状態２を有し、現在の状態が状態１であると仮定する。このメインスレッド２１０が状態１にある状態で外部からｒｃｖ＿ｍｓｇ（）関数によってイベント１を受信した場合、イベント関数１を実行して状態２に遷移し、入場関数２を実行する。メインスレッド２１０がこの状態２にあるときにイベント２を受信すると、イベント関数２を実行して状態１に戻り、入場関数１を実行する。一方、メインスレッド２１０が状態２にあるときにイベント３を受信すると、イベント関数３を実行し、状態２を維持する。

【００５７】

ここで、状態とは、イベントを待機する状態をいい、イベントとは、状態遷移のトリガとなる事象である。また、入場関数とは、状態遷移の際に実行される関数であり、イベント関数とは、発生したイベントに対して実行される関数である。

【００５８】

このような状態遷移モデルを利用してイベント処理を行う外部アプリ１１７を作成する際に、発生する全てのイベントを単一のモジュール、すなわち同じレベルの状態群で対応しようとすると、状態数の増大、一つの状態で処理するイベント数の増大が発生し、プログラムが複雑になり作業効率の低下、メンテナンス効率の低下という問題が生じる。

【００５９】

このため、本実施の形態では、外部アプリ１１７の全体で発生するイベント群から相互関係のあるイベントのみを処理する状態遷移モデルのモジュール（サブモジュール）を個別に作成している。そして、全体を制御する状態遷移モデルのモジュールからは、必要に応じて上記サブモジュールを関数呼び出しの形式で実行する。呼び出しを受けたサブモジュールは、必要に応じてさらに状態遷移モデルのサブモジュールの関数呼び出しを行う。

【００６０】

各モジュールは、外部アプリ１１７の全体で使用できるグローバル変数を使用せずに、起動時に共用するメモリを動的に確保することによって、同じモジュールが再帰的に呼び出されても正確に実行することが保証される。

【００６１】

図５は、ネスト構造を有する状態遷移モデルのモジュールからなる外部アプリ１１７の一例を示す説明図である。図５に示す例では、３つのネスト構造を有しており、イベント関数ｓｕｂ１（）によってネスト１のモジュールが呼び出され、イベント関数ｓｕｂ２（）によってネスト２のモジュールが呼び出されている。

【００６２】

次に、外部アプリ１１７のソースコードの具体的な記述について説明する。状態遷移モデルのモジュールで実行する入場関数、イベント関数において関数間で共用した変数がある場合、関数型の制約上、その変数を関数の引数として渡すことができない。また、このような変数をグローバル変数として定義すると、ネスト構造のモジュールやマルチスレッドの場合、予期せぬ変数の内容の変更が行われる可能性が高く、外部アプリ１１７の障害原因となる。

【００６３】

本実施の形態の外部アプリ１１７では、単一の状態遷移モデルのモジュールおよびネストしているモジュールで変数を共用可能としている。図６は、外部アプリ１１７のソースコードの定義部の一例を示す説明図である。

【００６４】

図６に示すように、外部アプリ１１７のソースコードでは、同一ファイルに記述された入場関数、イベント関数でグローバル変数のように扱う変数を集めて、共用変数として一つの構造体（ｄｅｍｏＣｏｍｍｏｎＰａｒｍ＿ｔ）を定義する。また、この共用変数に簡易にアクセスするためにＣＯＭＭＯＮ＿ＳＴＲＵＣＴを定義している。

【００６５】

また、図６の例では、外部アプリ１１７は、２つの状態ＤＥＭＯ＿ＳＴ＿ＩＤＬＥ，ＤＥＭＯ＿ＳＴ＿ＲＵＮを有し、各状態のイベント関数テーブルｅｖｅｎ

ｔＤｅｍｏＩｄｌｅ，ｅｖｅｎｔＤｅｍｏＲｕｎが定義されている。また、ｄｅｍｏＳｔＴｂｌ［］は、この状態遷移モデルのモジュールで使用する各状態の入場関数、イベント関数テーブルを定義した状態の順に記述している。また、ＴＨＲＥＡＤ＿ＮＯによってスレッド番号を指定することにより、このスレッド番号のスレッド内で有効な共用変数を使用することが可能となる。

【００６６】

図７は、外部アプリ１１７のソースコードの関数定義部の一例を示す説明図である。ｅｘｅｃＤｅｍｏＦｕｎｃ関数は、状態遷移モデルを起動する処理を行う。具体的には、共用変数の領域確保、設定の後、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）関数呼び出しを行う。図７の例では、スレッド番号ＴＨＲＥＡＤ＿ＮＯのスレッドで、ｄｅｍｏＳｔＴｂｌｅの状態情報テーブルに従い、状態番号ＤＥＭＯ＿ＳＴ＿ＩＤＬＥから起動し、ｐＣｏｍが示す領域を共用変数として使用することを指示して、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）を呼び出している。図７では、さらに図６で記述したイベント関数、入場関数の処理を記述する。

【００６７】

次に、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）の処理について説明する。図８は、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）の処理手順を示すフローチャートである。

【００６８】

状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）では、メッセージ受信を行うと、まず共用変数の領域割り当てを行う（ステップＳ７０１）。図９は、共用変数の領域割り当ての状態を示す説明図である。図９に示すように、本実施の形態ではスタックポインタｓｐの値がそのままネスト番号に相当するようになっており、各ネストのモジュールごと、すなわち各ネストのモジュールで状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）が実行されるたびに、共用変数群へのポインタがスタック領域に積まれるようになっている。従って、共用変数が下位モジュールで不用意に変更されることがなく、図９に示すように同じ変数名の共用変数を正確な値に維持しつつグローバル変数のように使用することができる。

【００６９】

次に、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）は、現在の状態のイベント関数テーブルを参照してその内容をチェックする（ステップＳ７０２）。そして、イベント関数テーブルに受信メッセージが定義されているか否かを判断する（ステップＳ７０３）。そして、受信メッセージが定義されている場合には、受信メッセージに対応するイベント関数を実行し（ステップＳ７０４）、さらに入場関数を実行する（ステップＳ７０５）。一方、イベント関数テーブルに受信メッセージが定義されていない場合には、イベント関数の実行は行わない。そして、このような処理をイベント関数テーブルの末尾に到達するまで繰り返し行う（ステップＳ７０６）。このようにして状態遷移モデルに基づいて作成された外部アプリ１１７の実行が行われる。

【００７０】

このように実施の形態１の複合機では、外部アプリ１１７が、複数のネスト構造のモジュールを含む外部アプリ１１７に実装され、各モジュールで共有する共用変数をモジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを２１５を有しているので、外部アプリ１１７を構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、複合機１００に外部アプリ１１７を搭載可能としながらも複合機１００の安定性を維持することができる。

【００７１】

（実施の形態２）

実施の形態１の複合機１００は、外部アプリ１１７が状態遷移モデルに基づいて作成されているため、状態に依存したイベント関数を実行するのみであったが、この実施の形態２の複合機１００では、外部アプリ１１７が、さらに状態に依存しない標準処理を行うものである。なお、実施の形態２の複合機１００および外部アプリ１１７のプロセス内部の構造は、実施の形態１と同様である。

【００７２】

実施の形態２の複合機１００では、ユーティリティライブラリの状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）が、標準処理を行うように構成されている。

【００７３】

図１０は、状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）の処理手順を示すフローチャートである。共用変数の割り当てから入場関数の実行までの処理（ステップＳ９０１～Ｓ９０６）は、実施の形態１の状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）と同様である。

【００７４】

本実施の形態の状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）では、さらに標準関数テーブルを参照し、その内容をチェックする（ステップＳ９０７）。ここで、標準関数とは、イベントを受信したときに現在の状態とは無関係、非依存に実行される処理の関数である。例えば、電源ＯＦＦのイベントに対する処理、アプリケーションのオーナ権限の問い合わせイベントに対し、オーナ権限の有無を通知する処理などがあげられる。なお、標準関数テーブルは、メッセージ（イベント）に対して実行される標準関数を定めたものであり、イベント関数テーブルと同様の構造を有している。

【００７５】

そして、標準関数テーブルに受信メッセージが定義されているか否かを判断する（ステップＳ９０８）。そして、受信メッセージが定義されている場合には、受信メッセージに対応する標準関数を実行する（ステップＳ９０９）。一方、標準関数テーブルに受信メッセージが定義されていない場合には、標準関数の実行は行わない。そして、このような処理を標準関数テーブルの末尾に到達するまで繰り返し行う（ステップＳ９１０）。このようにして状態遷移モデルに基づいて作成された外部アプリ１１７において、現在の状態に無関係に非同期にくるイベントに対する標準処理の実行が行われる。

【００７６】

このように実施の形態２の複合機１００では、状態遷移モデル起動関数は、現在の状態とは無関係に発生する非同期イベントを受信したときに行う処理を定めた標準関数を実行しているので、現在の状態とは無関係に生じる電源ＯＦＦなどのシステム関連のイベントなどに他のアプリに悪影響を与えずに対処することができ、複合機１００の安定性を向上させることができる。

【００７７】

【発明の効果】

以上説明したように、本発明によれば、プログラムを構成するモジュール間で共用変数を安全に使用することができ、変数の値に起因する障害を回避でき、情報処理装置に新規アプリケーションを搭載可能としながらも情報処理装置の安定性を維持することができるという効果を奏する。

【図面の簡単な説明】

【図１】

実施の形態１にかかる複合機の機能的構成を示すブロック図である。

【図２】

複合機のハードウェア構成を示す図である。

【図３】

外部アプリのプロセス内部の構造を示す説明図である。

【図４】

外部アプリが状態１と状態２を有する場合の状態遷移の説明図である。

【図５】

ネスト構造を有する状態遷移モデルのモジュールからなる外部アプリの一例を示す説明図である。

【図６】

外部アプリのソースコードの定義部の一例を示す説明図である。

【図７】

外部アプリのソースコードの関数定義部の一例を示す説明図である。

【図８】

状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）の処理手順を示すフローチャートである。

【図９】

共用変数の領域割り当ての状態を示す説明図である。

【図１０】

実施の形態２における状態遷移モデル起動関数ＳｔＭａｃｈｉｎｅ（）の処理

手順を示すフローチャートである。

【符号の説明】

１００　複合機

１０１　白黒レーザプリンタ

１０２　カラーレーザプリンタ

１０３　ＨＤＤ

１０４　ハードウェアリソース

１１０　ソフトウェア群

１１１　プリンタアプリ

１１２　コピーアプリ

１１３　ファックスアプリ

１１４　スキャナアプリ

１１５　ネットファイルアプリ

１１６　工程検査アプリ

１１７　外部アプリ

１２０　プラットホーム

１２１　汎用ＯＳ

１２２　ＳＣＳ

１２３　ＳＲＭ

１２４　ＥＣＳ

１２５　ＭＣＳ

１２６　ＯＣＳ

１２７　ＦＣＳ

１２８　ＮＣＳ

１３０　アプリケーション

１４０　コントロールサービス

２００　イメージライブラリスレッド

２１０　メインスレッド

２１１，２１２，２１４　メールボックス

２１３　イベント取得ハンドラ

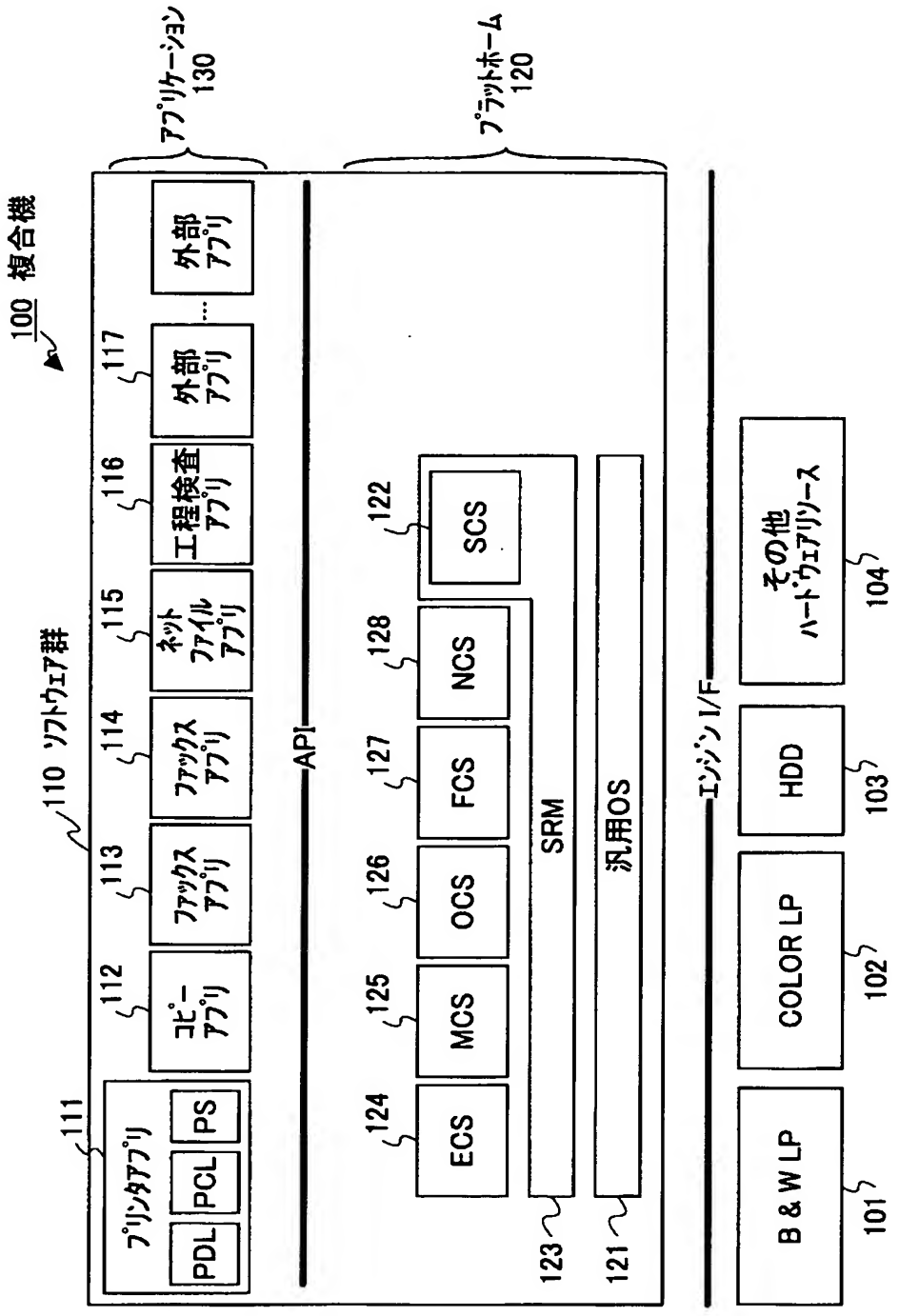
２１５　ユーティリティライブラリ

２２０　サブスレッド

【書類名】　図面

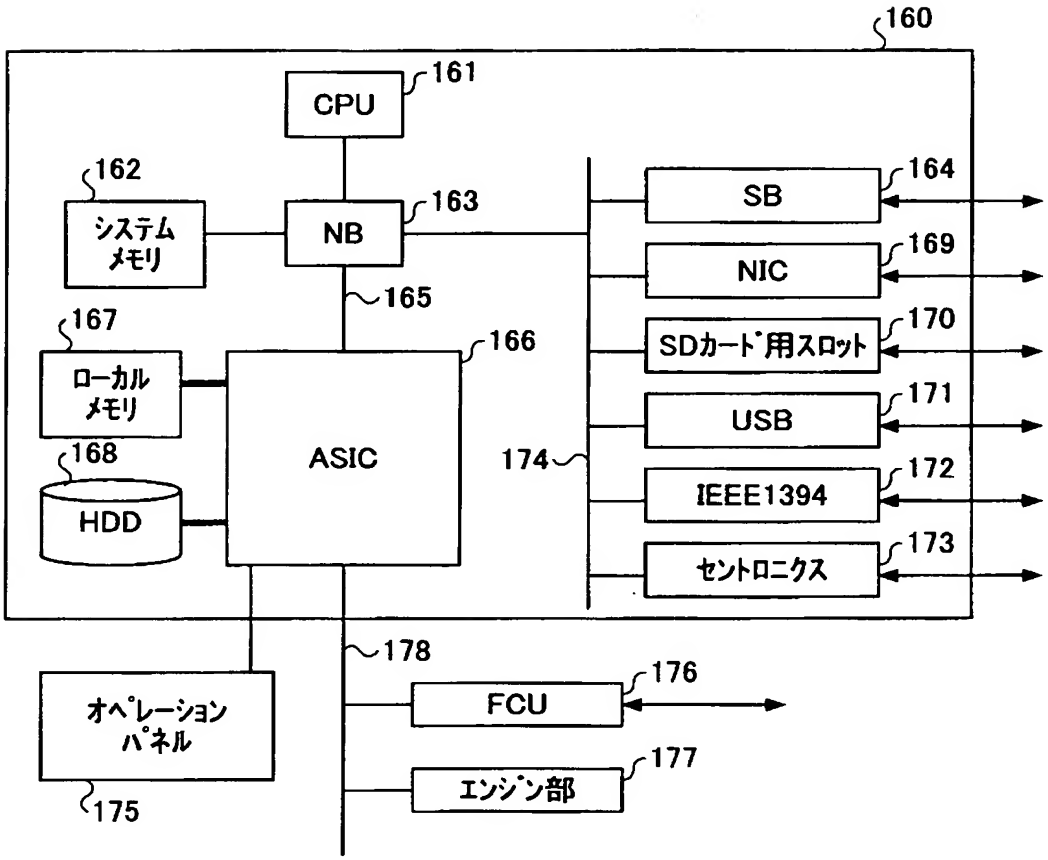
【図１】

実施の形態1にかかる複合機の機能的構成を示すブロック図



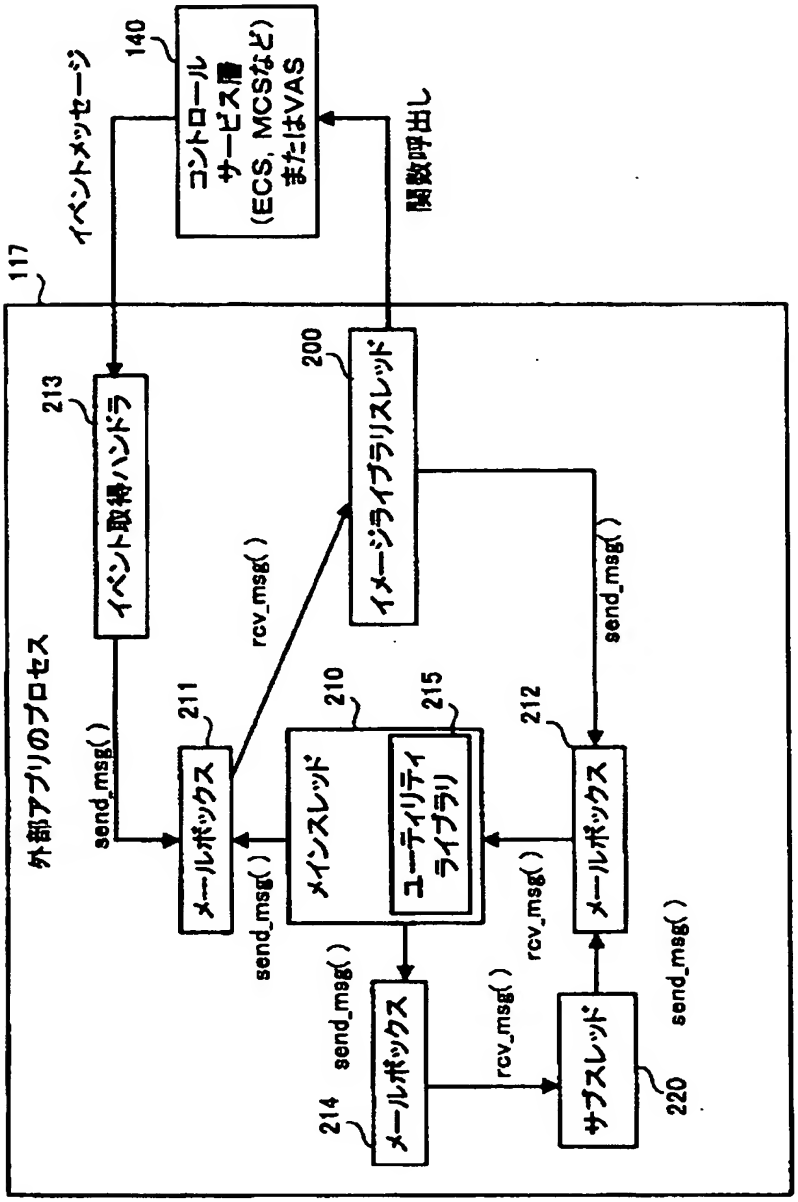
【図２】

複合機のハードウェア構成を示す図



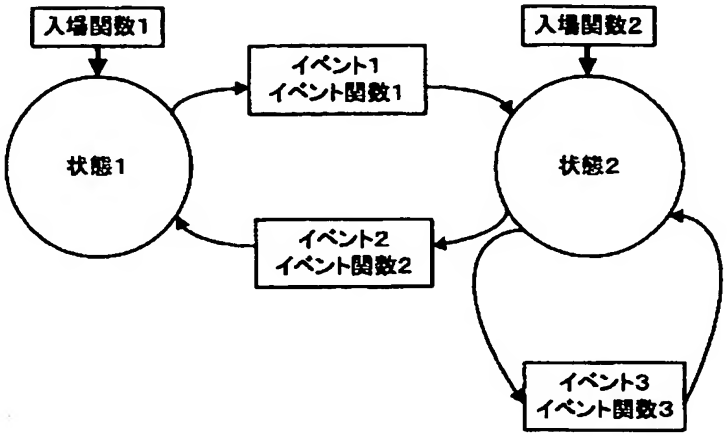
【図３】

外部アプリのプロセス内部の構造を示す説明図



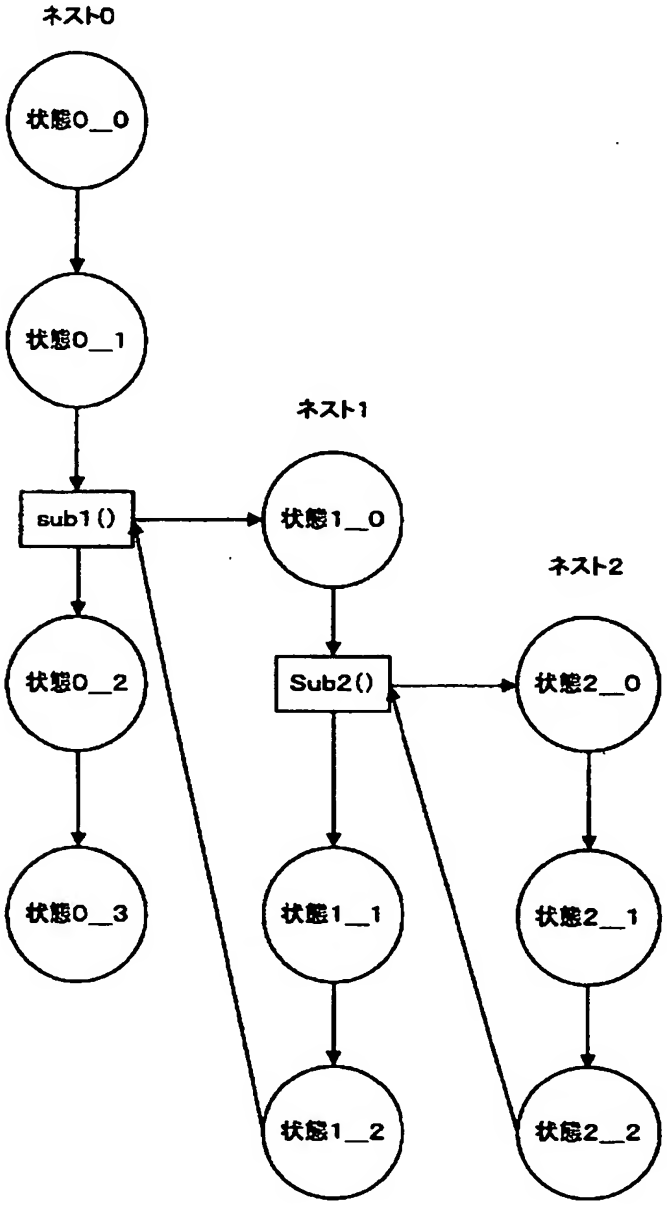
【図４】

外部アプリが状態1と状態2を有する場合の状態遷移の説明図



【図５】

ネスト構造を有する状態遷移モデルのモジュールからなる
外部アプリの一例を示す説明図



【図６】

外部アプリのソースコードの定義部の一例を示す説明図

```
#define  THREAD_NO 1              // スレッド番号:1
typedef struct {
        int      demoData;        //  共用変数の定義
} demoCommonParm_t;

#undef   COMMON_STRUCT
#define  COMMON_STRUCT demoCommonParm_t // 共用変数アクセスのための定義

enum DEMO_STATE{                  //  状態の定義
        DEMO_ST_IDLE,
        DEMO_ST_RUN
};

owner TypeFunc_t eventDemoIdle[] = //  状態DEMO_ST_IDLEのイベント関数テーブル
{
        {       1,      evDemoIdle },
        {       0,      0          }
};

owner TypeFunc_t eventDemoRun[] =  //  状態DEMO_ST_RUNのイベント関数テーブル
{
        {       2,      evDemoRun },
        {       0,      0         }
};

stateInfo_t demoStTbl[] =  //  状態情報テーブル
{
        {       enterDemoIdle,   eventDemoIdle },
        {       enterDemoRun,    eventDemoRun  }
};
```

【図７】

外部アプリのソースコードの関数定義部の一例を示す説明図

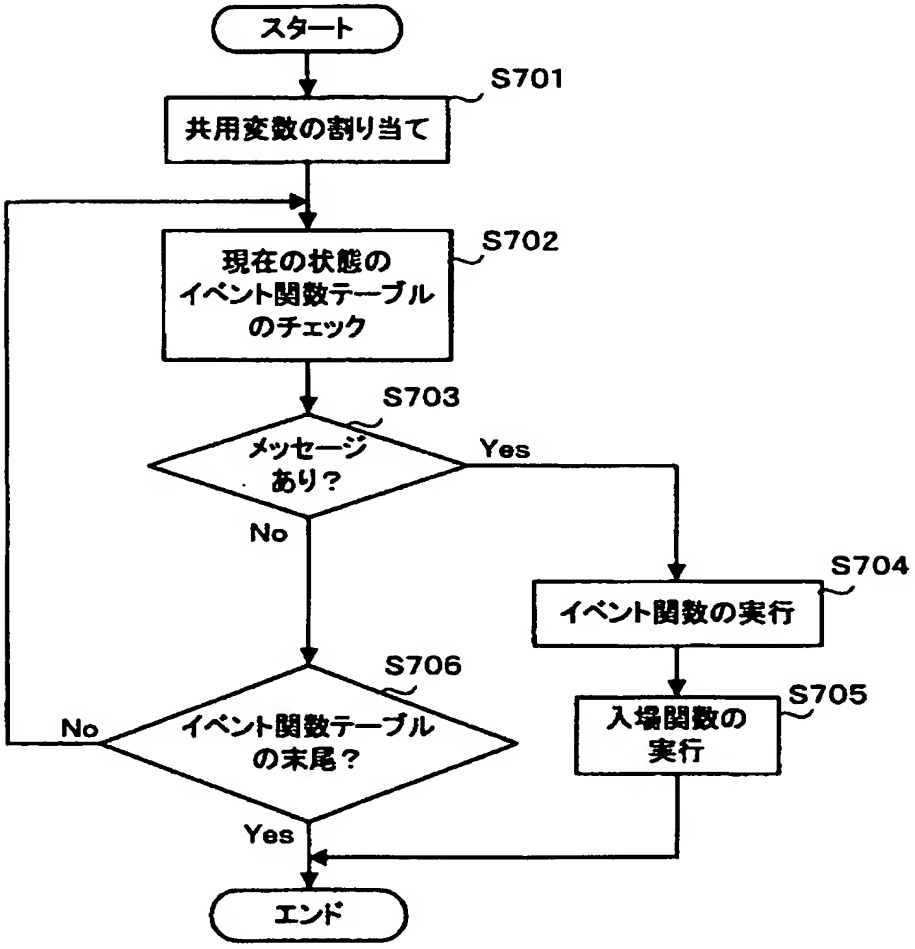
```
/*********************************************************
        execDemoFunc
*********************************************************/
void    execDemoFunc(void)
{
        int ret;
        COMMON_STRUCT *pCom;        // 共用変数構造体ポインタの定義
        pCom = (COMMON_STRUCT *)malloc(sizeof(COMMON_STRUCT));
                                    // 共用変数の領域確保
        pCom->demoData = 0;   // 共用変数の設定

        ret = StMachine(THREAD_NO, demoStTble, DEMO_ST_IDLE, pCom); // 起動
        free(pCom);                 // 共用変数の開放
}

/*********************************************************
        enterDemoIdle  入場関数
*********************************************************/
void    enterDemoIdle(void)
{
        ...
}
/*********************************************************
        evDemoIdle      イベント関数
*********************************************************/
void    evDemoIdle(void)
{
        ...
}
/*********************************************************
        enterDemoRun  入場関数
*********************************************************/
void    enterDemoRun(void)
{
        ...
}
/*********************************************************
        evDemoRun      イベント関数
*********************************************************/
void    evDemoRun(void)
{
        ...
}
```

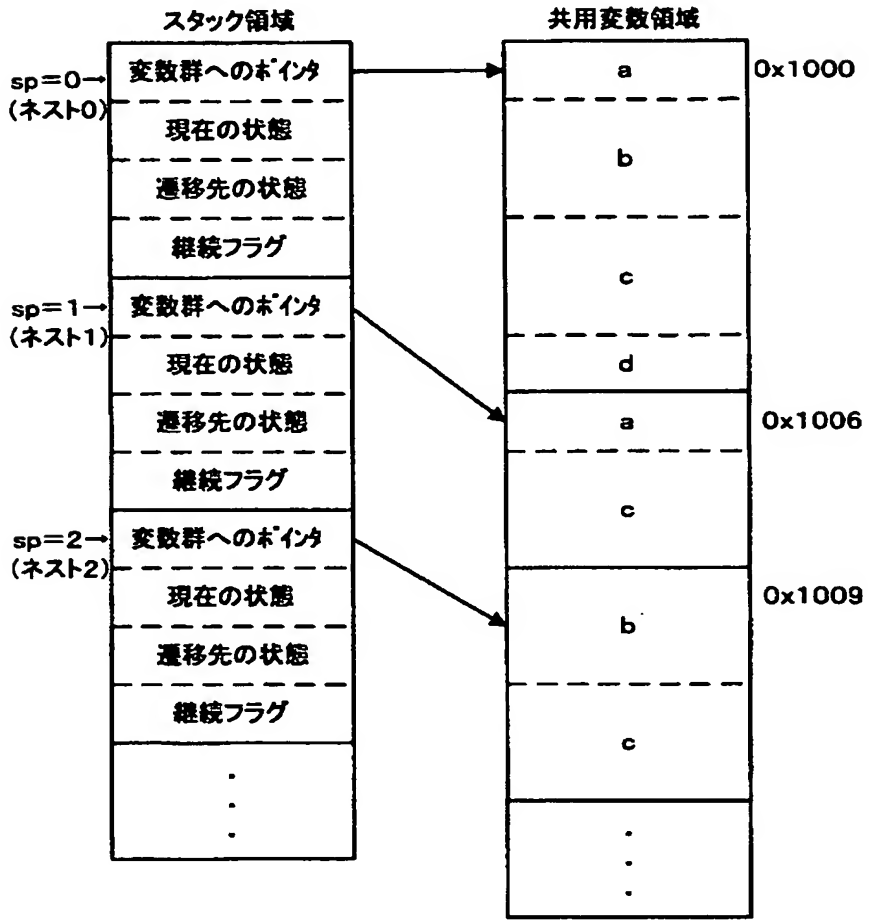
【図８】

状態遷移モデル起動関数
StMachine()の処理手順を示すフローチャート



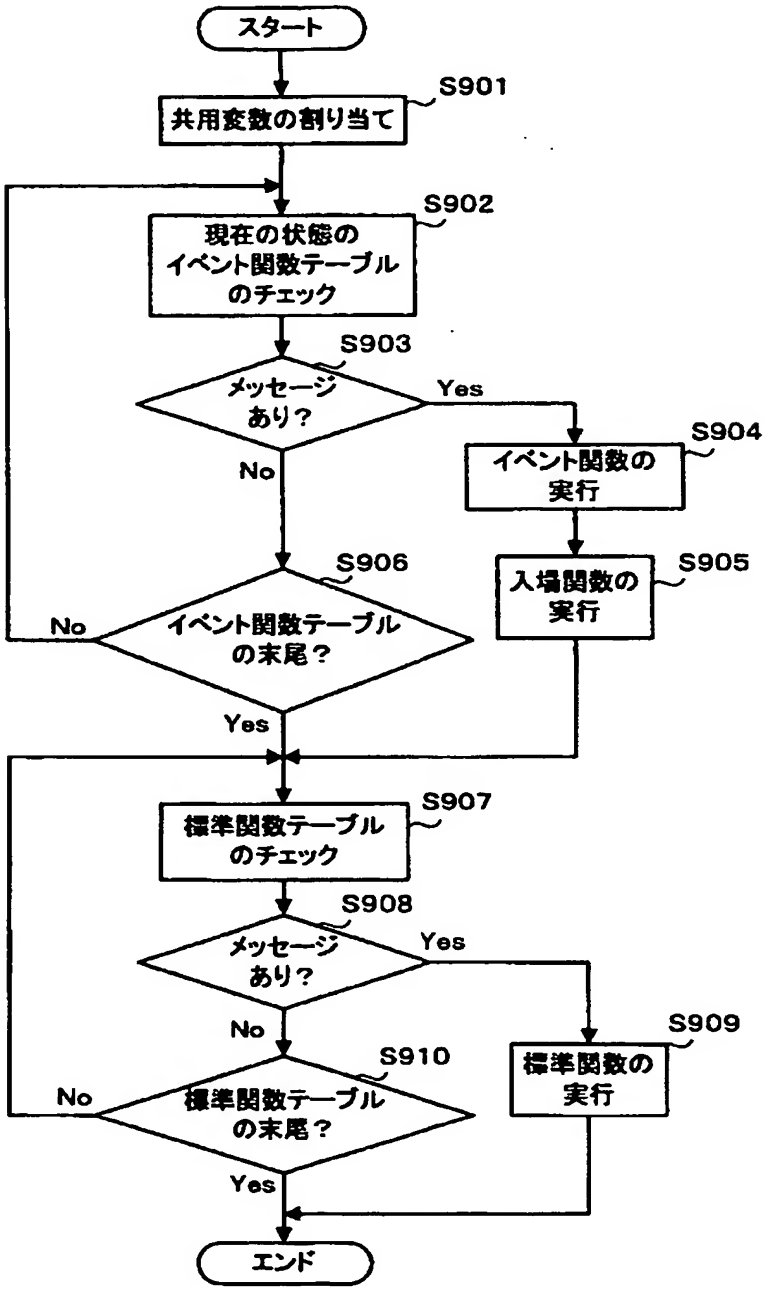
【図９】

共用変数の領域割り当ての状態を示す説明図



【図１０】

実施の形態2における状態遷移モデル起動関数StMachine()の処理手順を示すフローチャート



【書類名】　要約書

【要約】

【課題】　画像形成装置の安定性を保持しつつアプリケーションの追加を容易に行うこと。

【解決手段】　情報処理装置に、プログラムの取り得る状態と、前記状態下で前記プログラムに対して生じうるイベントと、前記イベントが前記状態下で生じたときに実行されるイベント関数と、前記イベント関数の実行後の遷移先の状態とによって動作が決定される状態遷移モデルに基づいた複数のネスト構造のモジュールを含む前記プログラムに実装され、各モジュールで共用する共用変数に対し前記モジュールごとに別個の領域に割り当てる状態遷移モデル起動関数が登録されたユーティリティライブラリを備えた。

【選択図】　図５

特願２００３－１９９９５０

出　願　人　履　歴　情　報

識別番号　　［０００００６７４７］

１．変更年月日　　１９９０年　８月２４日
［変更理由］　新規登録
住　所　東京都大田区中馬込１丁目３番６号
氏　名　株式会社リコー

２．変更年月日　　２００２年　５月１７日
［変更理由］　住所変更
住　所　東京都大田区中馬込１丁目３番６号
氏　名　株式会社リコー